**DMK220001**

**Denish Kalariya**

**Chatbot Project 2**

**Introduction**

This NLP project leverages traditional Machine Learning models alongside modern Natural Language Processing techniques to create a sophisticated chatbot. Our approach integrates models such as Naive Bayes (NB) and Support Vector Machines (SVM) with advanced structures including encoders, decoders, and attention mechanisms. The goal is to provide an interactive and intelligent system capable of understanding and responding to user inquiries about Python programming.

**Objective:**

The primary objective of our project is to develop a robust QA system that can pre-emptively resolve coding queries, thus streamlining the development process and increasing productivity. This system is designed not only to assist both novice and experienced programmers by providing immediate solutions to their coding issues but also to pave the way for future tools that enhance software development efficiency.

**Dataset:**

I have Used This Glaive Code Assistant dataset.

This Glaive Code Assistant dataset contains ~140k code problems and solutions designed to create intelligent Python code assistants. Structured in a QA format, this dataset contains real-world user questions worded for coding issues from the basics of data types to more complex object-oriented programming problems and features – with approximately 60% being Python. By using this dataset, developers can create automated systems that are able to accurately respond to the queries posed by users in any given environment. Creating an intelligent QA system could lead the way for new tools that solve user coding problems before they even arise, improving efficiency in development while simplifying their workflow. Whether you’re a beginner or advanced coder, this dataset has something of interest for all experience levels !This Glaive Code Assistant dataset contains ~140k code problems and solutions designed to create intelligent Python code assistants. Structured in a QA format, this dataset contains real-world user questions worded for coding issues from the basics of data types to more complex object-oriented programming problems and features – with approximately 60% being Python. By using this dataset, developers can create automated systems that are able to accurately respond to the queries posed by users in any given environment. Creating an intelligent QA system could lead the way for new tools that solve user coding problems before they even arise, improving efficiency in development while simplifying their workflow.

**Example of the Data:**

**A screenshot of a computer

Description automatically generated**

**System Description:**

**1. Encoder-Decoder Model (with Attention Mechanism)**

**Purpose**: This model is designed to generate contextually appropriate responses based on user inputs, making use of sequence-to-sequence learning typically employed in machine translation and chatbot applications.

I have taken only first 10000 rows to train the model due resource constraints.

**A screenshot of a computer

Description automatically generated**

A screenshot of a computer

Description automatically generated

**Key Components**:

Encoder: Processes the input text and converts it into a context vector.

Decoder: Uses the context vector to generate output text step by step.

Attention Mechanism: Enhances the model's ability to focus on relevant parts of the input during the decoding process, improving the relevance and specificity of responses.

Integration in Chatbot:

Role: Acts as the primary response generation engine. When a user query is received, this model processes the text to generate a coherent and contextually relevant response.

Data Flow: User inputs are preprocessed, tokenized, and fed into the encoder. The decoder then constructs a response, guided by the attention mechanism, which is delivered back to the user.

**2. Naive Bayes Classifier:**

**Purpose**: Utilized for classifying user queries into predefined categories, which can help in directing the query to the most suitable response mechanism within the chatbot.

**Key Components**:

* **TF-IDF Vectorization**: Converts text data into a format that can be efficiently processed by the Naive Bayes algorithm.
* **Naive Bayes Algorithm**: Predicts the category of the input query based on statistical inference.

**Integration in Chatbot**:

* **Role**: Acts as a preliminary filter to categorize user queries and decide if they can be answered directly with stored responses or if they need to be passed to the Encoder-Decoder model for generating a custom response.
* **Data Flow**: Incoming queries are vectorized and classified. Depending on the classification result, the query is either immediately answered or further processed by the Encoder-Decoder model.

**Support Vector Machine (SVM):**

**Purpose**: Utilized for classifying user queries into predefined categories, which can help in directing the query to the most suitable response mechanism within the chatbot.

**Key Components**:

* **TF-IDF Vectorization**: Converts text data into a format that can be efficiently processed by the Naive Bayes algorithm.
* **Naive Bayes Algorithm**: Predicts the category of the input query based on statistical inference.

**Integration in Chatbot**:

* **Role**: Acts as a preliminary filter to categorize user queries and decide if they can be answered directly with stored responses or if they need to be passed to the Encoder-Decoder model for generating a custom response.
* **Data Flow**: Incoming queries are vectorized and classified. Depending on the classification result, the query is either immediately answered or further processed by the Encoder-Decoder model.

**Specific NLP and ML techniques used:**

**1 . Encoder-Decoder Model (with Attention Mechanism)**

**Overview:** This segment implements an Encoder-Decoder architecture enhanced with an Attention mechanism, suitable for sequence-to-sequence learning, often used in machine translation and chatbot applications.

Key Components:

* Encoder: Uses LSTM (Long Short-Term Memory) units to process the input sequences and capture temporal dependencies. The encoder outputs a context vector representing the input sequence.
* Decoder: Also powered by LSTM units, it generates the output sequence step by step using the context vector and previous outputs.
* Attention Mechanism: Improves the model by allowing the decoder to focus on different parts of the encoder’s output for each step in the output sequence, thereby capturing nuances in longer sequences.

**Techniques**:

* Text Preprocessing: Includes converting characters from Unicode to ASCII, removing non-alphabetic characters, and handling contractions to clean and standardize the text.
* Tokenization and Padding: Converts text to sequences of integers and ensures that sequences are padded to a consistent length for modelling.
* Embedding: Transforms tokenized text into dense vectors that capture semantic meanings.
* LSTM with Dropout: Enhances the model's generalization by randomly dropping units (dropout) during training to prevent overfitting.

**2. Traditional ML Algo:**

**Naive Bayes Classifier:**

**Overview**: Utilizes a Naive Bayes model for classifying text, a popular statistical technique for NLP tasks like spam detection and sentiment analysis due to its simplicity and effectiveness.

**Techniques**:

* TF-IDF Vectorization: Transforms text into a meaningful vector of numbers based on the term frequency-inverse document frequency, which reflects the importance of words relative to the document and corpus.
* Model Training and Prediction: Employs the Naive Bayes algorithm, which assumes feature independence and calculates the probability of each category based on the Bayes theorem.

**Support Vector Machine (SVM):**

**Overview**: Implements an SVM for text classification. SVM is a powerful, supervised machine learning model that is effective in high-dimensional spaces and ideal for binary classification tasks.

**Technique**s:

* Text Cleaning and Lemmatization: Involves removing special characters and converting words to their base or dictionary form, reducing complexity and variability in the text.
* SpaCy for Tokenization and Lemmatization: Leverages the SpaCy library to process text for tokenization and lemmatization, helping to further refine the text for analysis.
* TF-IDF Vectorization: Similar to the Naive Bayes section, it prepares text data by converting it into TF-IDF vectors.
* SVM Training: Trains an SVM classifier with a linear kernel to distinguish between different classes based on the decision boundaries defined in the high-dimensional feature space.

**3. ChatBot:**

**Overview**: This part integrates user interaction and personalization into the chatbot, enhancing user experience by tailoring responses and maintaining a context of interaction.

**Techniques**:

* Personal Data Handling: Manages and utilizes user-specific data to personalize interactions, improving engagement.
* Pattern Matching: Uses regular expressions to identify user preferences (likes and dislikes) and other personal details from the conversation, which can influence the chatbot’s responses.
* Used NER to extract the information from the User Input
* Dynamic Response Generation: Based on the processed input and personal data, the chatbot generates and outputs personalized responses.

**Data Cleaning for Different Methods:**

**1. Unicode Normalization**

**Function Used**: **unicode\_to\_ascii(s)**

* **Purpose**: Converts Unicode characters to ASCII, primarily by removing diacritics from characters. This is important for standardizing text data and avoiding issues with character encoding.
* **Method**: Utilizes Python’s **unicodedata.normalize** function to decompose characters into their base characters and diacritics, then filters out diacritic marks using a list comprehension.

**2. Lowercasing and Stripping Whitespace**

**Applied In**: **preprocess\_text(text)**

* **Purpose**: Converts all text to lowercase to reduce complexity and variability, making the text uniform. Stripping removes any leading or trailing whitespace, cleaning up the input.
* **Method**: Text strings are modified with the **.lower()** method for case normalization and **.strip()** method to remove extra spaces.

**3. Removing Non-Alphabetic Characters**

**Applied In**: **preprocess\_text(text)**

* **Purpose**: Cleans the text of punctuation, special characters, and other non-essential elements that might confuse the models.
* **Method**: Uses regular expressions (**re.sub**) to replace non-word characters with spaces.

**4. Exclusion of Words Containing Numbers**

**Applied In**: **preprocess\_text(text)**

* **Purpose**: Removes words that contain digits, which are generally irrelevant for natural language understanding tasks.
* **Method**: Another application of **re.sub**, which filters out any token that includes digits, ensuring that only purely textual data is processed.

**5. Handling Contractions**

**Applied In**: **preprocess\_text(text)**

* **Purpose**: Expands contractions (e.g., changing “can’t” to “cannot”) to standardize text and improve the matching process during tokenization.
* **Method**: A dictionary of contractions is used, and each word in the text is replaced with its expanded form if it exists in the dictionary.

**6. Removing Stopwords**

**In SVM and NB Sections**

* **Purpose**: Stopwords (common words like 'the', 'is', 'at') are typically removed because they offer little value in the way of understanding the intent of a question or response.
* **Method**: Utilizes NLTK’s list of English stopwords and filters out these words from the processed text.

**7. Lemmatization**

**Applied In**: **lemmatize\_text(text)** in SVM section

* **Purpose**: Reduces words to their base or dictionary form (lemma) to treat different forms of a word as the same, reducing the complexity of the model’s input space.
* **Method**: Utilizes SpaCy’s linguistic annotations to convert each token to its lemma.

**8. Text Vectorization (TF-IDF)**

**In SVM and NB Sections**

* **Purpose**: Converts text into a numerical format that machine learning algorithms can process, by weighing the terms based on their frequency and inverse document frequency across the corpus.
* **Method**: **TfidfVectorizer** from scikit-learn transforms text into a sparse matrix of TF-IDF features.

**9. Named Entity Recognition (NER) Usage**

**Purpose**: Named Entity Recognition (NER) is used to identify and classify named entities mentioned in text into pre-defined categories such as names of people, organizations, locations, expressions of times, quantities, monetary values, percentages, etc. In the context of your chatbot, NER is used to enhance user interaction by recognizing personal details which can then be used to tailor the chatbot’s responses.

**Implementation Details:**

* Library Used: SpaCy
* Model: "en\_core\_web\_sm"
* Function: extract\_and\_update\_user\_data(text, user\_data)

**Operational Steps:**

1. Text Processing:
   * The text input from the user is processed through SpaCy's NLP model. This model has been pre-trained on a variety of text corpora and is capable of performing several NLP tasks, including NER.
2. Entity Extraction:
   * As the text is processed, SpaCy identifies entities and classifies them into categories like PERSON (names), ORG (organizations), LOC (locations), and DATE (dates).
   * Each identified entity is tagged with its corresponding category, which the chatbot can use to understand more about the user’s context or intent.
3. Utilizing Extracted Entities:
   * The extracted entities are used within the chatbot to personalize the conversation. For example, if a user mentions their name or a place, the chatbot can acknowledge this in subsequent responses, thereby creating a more engaging and personalized interaction.
   * Personal details extracted via NER are stored in a user-specific data structure (user\_data). This information may include the user's name, associated locations, important dates, etc.
4. Response Personalization:
   * The chatbot uses the stored personal information to tailor its responses. For instance, if a user mentions their name, the chatbot might use this name in future interactions to create a more personalized and conversational experience.

Top of Form

**A diagram of logic:**

A black background with white ovals

Description automatically generated

**Sample Dialog of Interaction:**

**For Naïve Bayes:**

A screenshot of a computer

Description automatically generated

**For SVM:**

**A screenshot of a computer

Description automatically generated**

**Encoder-Decoder Model (with Attention Mechanism)**

A white background with black text

Description automatically generated

**Appendix for User-Model :**

The JSON structure represents a user profile for a chatbot, detailing the user's preferences and personal information, which the chatbot can use to personalize interactions:

* **name**: A string representing the user's name, which is used to personalize the interaction and file naming for saving user data.
* **likes**: An array of strings representing the things that the user likes. The chatbot can use this information to relate to the user and bring up topics that the user is interested in.
* **dislikes**: An array of strings indicating what the user dislikes. This information helps the chatbot avoid topics or things that might displease the user.
* **personal\_info**: An object that stores structured personal information about the user:
  + **org** (organization): A string indicating the name of an organization with which the user is associated, in this case, "University of Texas". The chatbot can reference this in conversation for a more personalized experience.
  + **gpe** (geopolitical entity): A string denoting a geopolitical location, "Dallas" in this case. This information can be used to discuss location-specific topics or tailor the conversation to the user's locale.

The chatbot can reference and update this structure during the conversation to both reflect and react to new information provided by the user.

Example :

{

"name": "Denish",

"likes": [

"coffee",

"tea"

],

"dislikes": [

"tea"

],

"personal\_info": {

"org": "University of Texas",

"gpe": "Dallas"

}

}

* **"name": "Denish"**
  + This key-value pair holds the name of the user, which the chatbot can use to address the user directly, making interactions feel more personal and engaging.
* **"likes": ["coffee", "tea"]**
  + This is an array of strings under the key "likes". It represents the things that the user, Denish, enjoys. In this example, Denish has expressed a liking for "coffee" and "tea". This information can be used by the chatbot to make conversation about these beverages or to suggest related topics.
* **"dislikes": ["tea"]**
  + Under the "dislikes" key is an array that lists things the user does not enjoy. Interestingly, "tea" appears here as well as in the "likes" array. This inconsistency could be an input error or might suggest that Denish has a complicated relationship with tea, such as liking certain types but disliking others. For a chatbot, this would require further clarification with the user to understand their preferences better.
* **"personal\_info": { "org": "University of Texas", "gpe": "Dallas" }**
  + The "personal\_info" object contains structured data about the user's affiliations and location.
    - **"org": "University of Texas"**: Indicates that Denish is associated with the University of Texas, which could be an educational institution or a workplace. This can be used by the chatbot to discuss academic or work-related topics.
    - **"gpe": "Dallas"**: Stands for Geopolitical Entity, and here it specifies that Denish is located in or associated with Dallas. This information might be used to customize the chat experience with local news, weather, or events.

Usage in Chatbot Interaction:

* Customization: The chatbot can use these preferences to tailor dialogues, such as discussing programming topics related to Java or referencing events at the University of Texas.
* Recommendations: Based on his likes, the chatbot could recommend coffee brands, Java tutorials, or local events in Dallas.
* Engagement Strategies: To engage Denish effectively, the chatbot might initiate conversations by mentioning updates or news related to Java programming or inquire about his experiences at the University of Texas.

Implications for System Development:

* Enhanced Personalization: This model exemplifies how detailed user profiles can facilitate deeper personalization, enhancing user experience by making interactions more relevant and engaging.